Grid-based Collaboration in Interactive Data Language Applications

Minjun Wang
EECS Department, Syracuse University, U.S.A
Community Grid Laboratory, Indiana University, U.S.A
501 N Morton, Suite 222, Bloomington IN 47404
minwang@indiana.edu

Geoffrey Fox
Community Grid Laboratory, Computer Science Department, School of Informatics and Physics Department, Indiana University, U.S.A
gcf@indiana.edu

Marlon Pierce
Community Grid Laboratory, Indiana University, U.S.A
501 N Morton, Suite 224, Bloomington IN 47404
mpierce@cs.indiana.edu

Abstract

Interactive Data Language (IDL) is an array-oriented data analysis and visualization application, which is widely used in research, commerce, and education.

It is meaningful to make user IDL applications collaborative between computers over networks, using a common message broker as the underlying communication system.

In order to achieve the global collaboration, we have brought together in the research a Grid-based Collaboration paradigm, a Shared Event model, different implementing structures, methodologies and technologies. We have succeeded in our prototype codes, and we are currently working on a real life IDL application package to make it collaborative. At the same time, we are trying to find better structures and methods for the collaboration in general user IDL applications.

1. Introduction

Interactive Data Language (IDL) is an array-oriented data analysis and visualization application, which is widely used in research, commerce, and education [1, 2]. Its application areas include engineering, medical physics, astronomical and space science, earth science, etc.

It offers rapid interactive data analysis and visualization, a programming environment, and end user applications.

IDL is available for Windows, UNIX, Linux, Macintosh and VMS platforms and Operating Systems. This high availability facilitates data analysis and visualization in multi-platform environment, and ensures high code portability among platforms and systems.

People from different categories around the world have developed and been using diverse IDL applications in their respective areas.

Also, users worldwide are continually contributing to Internet-based IDL libraries [3], and they are freely available.

It is contributing and meaningful to make IDL applications collaborative between computers of same or different platforms, using a common message broker – such as NaradaBrokering Message Service – as the underlying communication system.

In today's Information revolution society, collaboration is becoming more and more important. It means breakthroughs in new abilities, which otherwise would be hardly possible; it means achievements in the advances of science and technology; it also means great contributions to economy.

We design the overall structure of the collaborative IDL applications to consist of a type of Master (or Master Client) and a type of Participant (or Participating Client) using small text event messages for the communication between them. During a session, the Master captures events in its process, deal with them, and send the event messages to the participant for rendering the displays in the participant's process, so that both of them can share the screen displays simultaneously. There can be multiple participants working with the Master concurrently and independently. We use NaradaBrokering Message Service [4] for the message communication. The RSI IDL software should be installed on both the hosts of the Master and the Participant, and if files are needed in a session, they are deployed beforehand on the same directories on the hosts.

We research and explore this area by proposing and using:

- A Grid-base Collaboration paradigm, in which Shared Event Model as messenger, and Peer-to-Peer Grid computing [5] as basis.
- Different Implementation Structures for the collaborative IDL applications -- Notifying Structure and Polling Structure.

We illustrate the mechanisms, methodologies and technologies used in each structure, and analyze their strengths and limitations in the context of applications.

We have developed event-driven GUI programs and made them collaborative through networks to demonstrate in the Notifying and Polling structures. We are working on a real IDL application package – ReviewPlus [6] from General Atomics (USA) [7], which is a general-purpose data visualization tool – and trying to make it collaborative by using the Polling structure. We will describe the development and special issues in the implementation in this paper.

We think our work will contribute to Grid-based Collaboration in Interactive Data Language Applications.

2. Grid-based Collaboration Model

We use a Grid-based Collaboration Model in the design and development of the collaborative Interactive Data Language (IDL) applications, as shown in Figure 1.

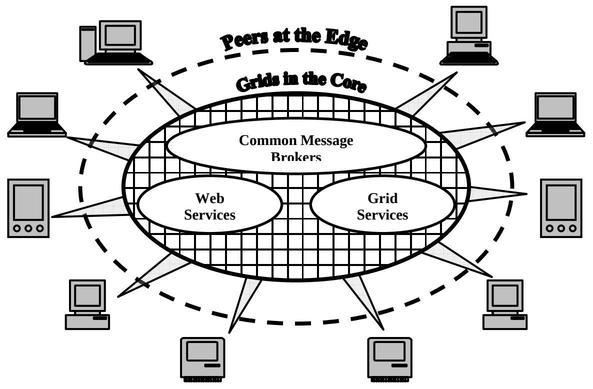


Figure 1. A Grid-based Collaboration Model

In this model, there are two categories of computing – Grid computing and Peer-to-Peer computing.

Grid computing is the basis; it largely comprises stable, formal, and efficient high-functionality services like Web Services, Grid Services, Common Message Brokers, etc., which are deployed as Grids on structured, well-organized and powerful supercomputers. They are in the core of the model.

Peer-to-Peer computing is the interface to this world; it offers user-friendly, convenient, intuitive and easy accessible applications and services such as the popular commodity software used daily and everywhere. They are installed on a variety of personal devices, such as desktops, laptops, PDAs, smart phones, etc. They are at the edge of the model.

The infrastructure of Networks and the Internet ties up and correlates the two computing categories. It enables Peer-to-Peer Grids computing to be a trend, which harnesses the advantages of the two categories so that they complement each other, which also brings new opportunities and challenges to computing in all.

Grid computing offers robust, structured, security services that scale well in pre-existing hierarchically arranged enterprises or organizations; it is largely asynchronous and allows seamless access to supercomputers and their datasets.

Peer-to-Peer computing is more convenient and efficient for the low-end clients to advertise and access the files on the communal computers; it is more intuitive, unstructured, and largely synchronous.

In our design and development of the collaborative IDL applications, we realize the Peer-to-Peer Grids computing idea. We deploy the Narada Message Broker as a Grid and use it for message communication between the Master and Participants of the applications; and we deploy the Master and Participants as Peers at the edge and make them collaborate on events.

3. Shared Event Model

We use a Shared Event Model in the communication between Peers. In this model, small text event messages are transmitted via the Grids of common message brokers and used to coordinate the operations between the peers so that they can cooperate concurrently and share the output screen simultaneously.

In our design of the collaborative IDL applications, one type of the Peers is Master Client, another type is Participant. During a session, the Master captures events in its process, deals with them, and sends the event messages to the participant for rendering the displays in the participant's process, so that both of them can share the screen displays simultaneously. There can be multiple participants working with the Master concurrently and independently. We use Narada Message Broker as the Grid for the message communication. The RSI IDL software should be installed on both the hosts of the Master and the Participant, and if files are needed in a session, they are deployed beforehand on the same directories on the hosts. This deployment guarantees the access of the files is correct on the hosts under the control of event messages.

There are a variety of primitive widgets in IDL, such as Button, Slider, Text field, Draw area, etc. The event structure for each widget is different; each one contains state information specific to that widget, e.g., flags and values. However, there are three common items in all the event structures, they are: ID, TOP, and HANDLER. They are long integers and the first three items in the structure.

1. ID is the widget ID number of the widget that generates the event.
2. TOP is the widget ID number of the top-level base that contains all the widgets.
3. HANDLER is the widget ID number of the widget that is associated with an event handler.

For instance, the event structure for BASE widget is:

{WIDGET_BASE, ID:0L, TOP:0L, HANDLER:0L, X:0L, Y:0L}

Where X is the width of the base, and Y is the height.

On the Master, the client captures the event, gets the event structure, and packages the information from it into a delimited string, as in {widget_base|id 0|top 0|handler 0|x 0|y 0}, with possibly other information like session, source, destination, etc., and sends the result message string to Narada message broker for broadcasting to participants. This is a serialization process.

On the participant, the client parses the received message string, gets the different part of the delimited information, and rebuilds the IDL event structure by interpreting the sub-string sections like “id 0” to corresponding IDL types. This is a de-serialization process.

The constructed event structure is then used as a parameter for its event handler which is invoked by the participant client programs to generate the same event results as that happened on the Master client.

4. Notifying Structure

We have developed and made a simple IDL GUI application collaborative between the Master and Participating clients based on the Notifying structure.

The master client displays a GUI containing a lot of button widgets which represent JPEG images.

When a user clicks a button:

- The corresponding image displays in IDL environment.
- The master client captures the event and sends the message to NaradaBrokering to broadcast to participating clients for rendering.

The participant receives the event message broadcasted from NaradaBrokering, and renders the display as that of the Master in its own IDL environment. There can be multiple instances of participant clients. The interface and IDL display on both the Master and Participant are shown in Figure 2.

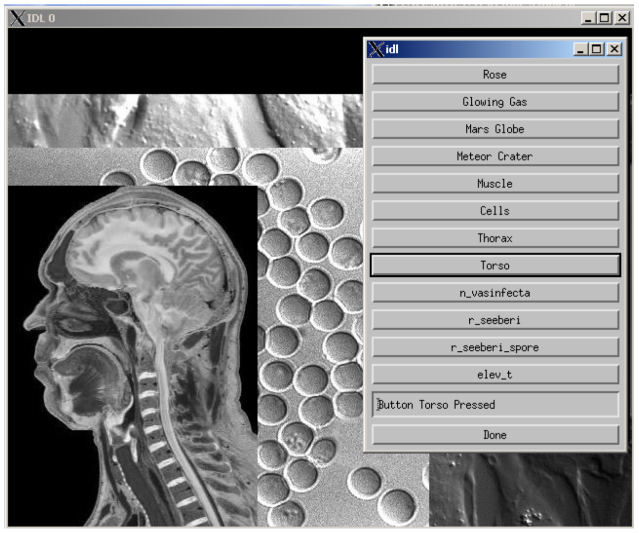


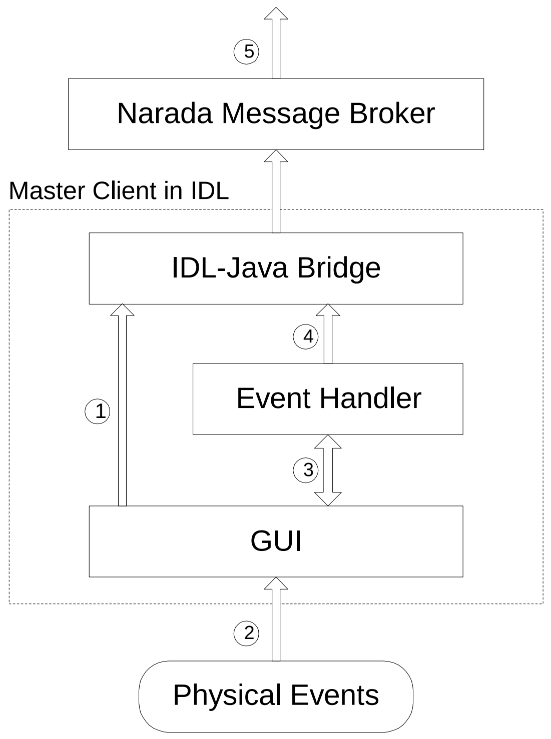
Figure 2. The Interface and Display from a Collaborative IDL Application

In this structure, we just naturally followed the nature and technologies in IDL and NaradaBrokering (written in Java) required being collaborative. The technologies for this issue include IDL-Java Bridge, Callable IDL, Shared Library, Java Native Language (JNI) [8], Subscribe/Notify mechanism. Whenever NaradaBrokering receives an event message from the Master client, it will be *Notifying* the participating clients through its method *onMessage()*, which in turn gets the message and invokes all kinds of IDL routines accordingly to render the display.

More specifically, the Master client is written in IDL programs. It consists of a GUI building and managing part, and an event handling part.

- It captures an event and gets the event message in an event handler whenever a user clicks a button in the GUI.
- It makes use of the IDL-Java Bridge, calls methods in a Java program to connect to NaradaBrokering, and sends the event message over there for broadcasting to participants.

This process is elaborated in Figure 3.



1 GUI calls methods of Narada Message Broker via IDL-Java Bridge and connects Master client to the Broker

2 User interacts with GUI through Physical Events (mouse clicks, keyboard strokes) to control session

3 Event Handler associates to GUI; GUI Notifies Event Handler whenever a GUI event occurs

4 Event Handler processes event and sends message to Narada Message Broker via IDL-Java Bridge

5 Narada Message Broker broadcasts message to all subscribed clients

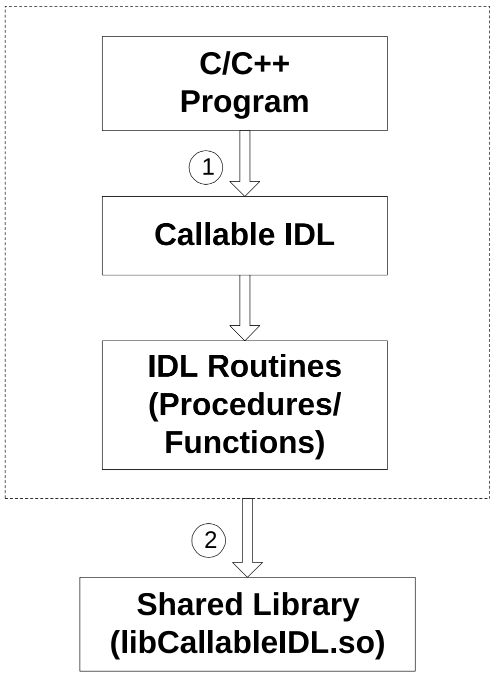
Figure 3. The Mechanism of Master Client

The participant is written in Java programs.

- It connects to NaradaBrokering and receives event messages from it.
- The Java program controls the rendering process according to the event messages it receives.
    - It makes use of the Callable IDL technology and JNI technology.
    - It calls the IDL routines (procedures or functions) for the rendering.
    - In order to do that, it has to call the IDL routines through a C program, in other words, that C program calls IDL routines directly through Callable IDL technology.
    - A shared library (libCallableIDL.so) is generated from the C program, and the Java program calls the native functions in the shared library through JNI.

This process is elaborated in Figure 4 and 5.

This way, it renders the images simultaneously with the master client.


C/C++ Program
1
Callable IDL
IDL Routines (Procedures/ Functions)
2
Shared Library (libCallableIDL.so)

1 C/C++ program calls IDL routines through Callable IDL

2 A Shared Library is generated from C/C++ program which calls IDL Routines

Figure 4. Generating of a Shared Library

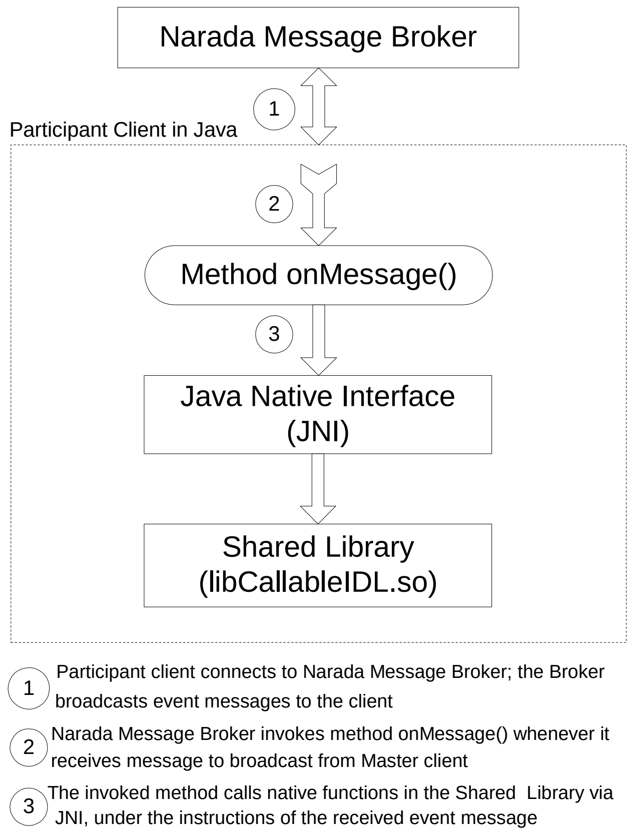

Narada Message Broker
1
Participant Client in Java
2
Method onMessage()
3
Java Native Interface (JNI)
Shared Library (libCallableIDL.so)
1 Participant client connects to Narada Message Broker; the Broker broadcasts event messages to the client
2 Narada Message Broker invokes method onMessage() whenever it receives message to broadcast from Master client
3 The invoked method calls native functions in the Shared Library via JNI, under the instructions of the received event message

Figure 5. The Mechanism of Participant Client

This structure works just fine for our small demonstrating collaborative IDL GUI applications, it follows the nature of the IDL and Narada systems, and it builds on top of those systems and just makes use of their technologies and functions in achieving collaboration without changing anything in either of them. In other words, all of the design and programming is just within our collaborative applications. This is architectural abstraction thinking and complies with object oriented design in systems level.

There is a limitation in this structure, that is, the types of the programs for the master and participant clients are different. The program for the master client is in IDL (.pro) language, while that for the participant client is in Java (.java), calling native functions in a shared library through JNI. The shared library in turn is generated from a C program, which calls IDL routines directly through Callable IDL technology. In other words, all the functionality of the IDL routines is compiled into binary and put in the shared library (.so). This is not good enough for large application development, like ReviewPlus, which is huge itself, and also refers to many (if not huge) other IDL routines in other IDL applications such as MDSplus [9].

This limitation implies:

1. It is complicated and inconsistent in the codes between master and participant clients. They look totally different things and there is no similarity.
2. The time and efforts in developing would be doubled, if not more – one for master, and one for participant, two different kinds.
3. It is error-prone in programming, debugging and testing, bringing different technologies and environments together.

5. Polling Structure

Now that we have succeeded in making a simple IDL application collaborative, we begin to think further and ask a question: Is it possible to develop the codes for both mater and participant clients in pure IDL and make the codes for master and the codes for participant as same as possible? If we succeed in it, we can overcome the limitation and shortcomings mentioned in the Notifying structure, and at the same time, we can simplify the collaboration system and make the codes consistent and clear; we can save significant time, effort and cost in software development and maintenance. Thereof, we can really make Grid-based collaboration for large IDL applications such as ReviewPlus practical and feasible.

We have tried and succeeded in our simple GUI IDL collaborative applications for this using the Polling Structure. There is a trade-off: in order to achieve this, we have to change some parts of the underlying systems in some cases, thus suffering some design abstraction; in our case, we changed an interface to the underlying Narada Message Broker.

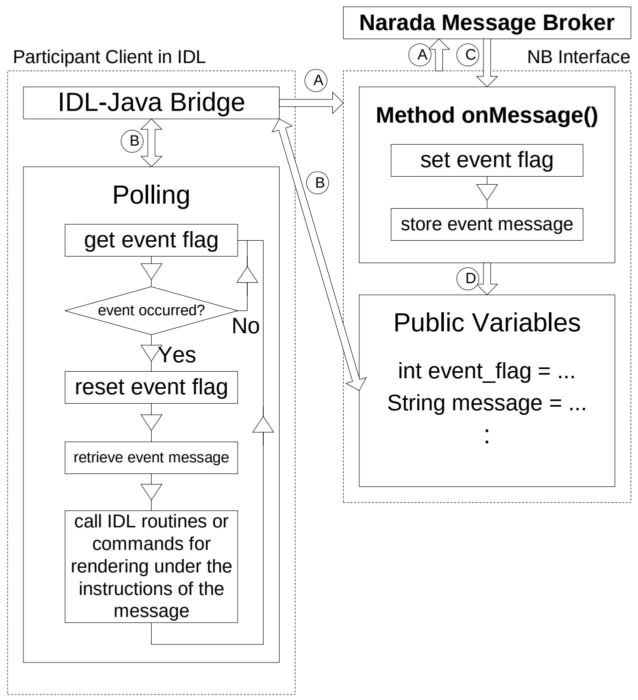
The Polling Structure works like this:

Both the master and participant clients make use of the IDL-Java Bridge to connect to NaradaBrokering and communicate with it. The methods used in the Bridge belong to IDL.

As before, the master captures events and sends event messages to NaradaBroking, which then broadcasts them to all participating clients, as in Figure 3. In a Java class, which is an interface to NaradaBrokering, and which the participating clients codes instantiate and make use of, we add public global variables for event change flag and event message, and make a notification related method *onMessage()* update them whenever the Broker broadcasts event messages to the clients. The update includes setting event flag and storing event message in the variables.

The participating client code now has an instance of the Java class; it is constantly testing, or *Polling*, the instance variable – event flag. If it finds the flag is set, it resets the flag and retrieves the event message from the event message instance variable. It then follows the instructions of the message to execute different parts of the IDL programs to do the rendering.

This process is elaborated in Figure 6.



(A) Participant client connects to NB by calling methods of NB interface via IDL-Java Bridge

(B) The client accesses the public variables of NB interface by calling the Bridge's methods getProperty() and setProperty()

(C) The Broker invokes method onMessage() of NB interface when it has event message to broadcast

(D) Method onMessage() then accesses the public variables of NB interface by setting event flag and storing message

Figure 6. The Mechanism of Participant Client in Polling Structure

This way, the Polling structure makes the collaboration working. It has advantages in working with large IDL applications. We are using it in the design and implementing of the collaborative ReviewPlus applications. The interface and display of ReviewPlus is shown in Figure 7.

As in IDL widget programming, the structure of ReviewPlus consists of two parts; one is the widgets definition part, the other is the event handling routine part. All the required widgets in the application are defined and realized in the former, and the event handlers are contained in the latter. The event handling part is put first in a program unit, and the definition part follows.

People specify the even function or procedure to be called when a widget is invoked by using the keywords EVENT_FUNC or EVENT_PRO in the definition of the widget. This way, when the widget is invoked, the corresponding event handler is called to process the event.

For example, in ReviewPlus, there is a piece of code:

```
mEdit = Widget_Button(menubase, Value='Edit')
x = widget_button(mEdit, value='Set Signals', $
event_pro='ReviewPlus_SignalDialog_event')
```

This is for the item “Set Signals” on menu “Edit”. When this widget fires an event, the event handler “ReviewPlus_SignalDialog_event” is invoked.

Both the Master and Participant are developed on the basis of ReviewPlus and make use of its codes as much as possible.

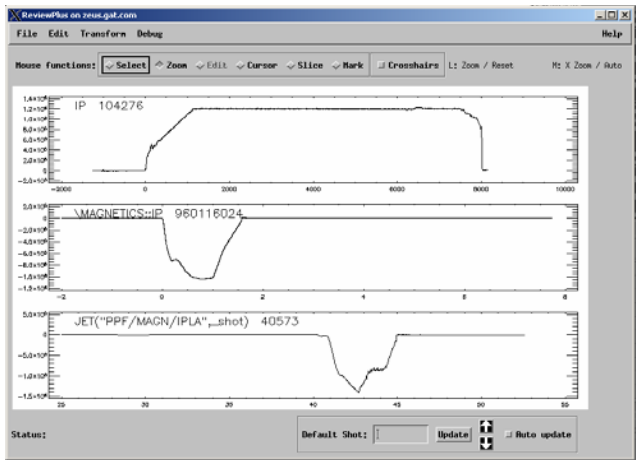


Figure 7. The Interface and Display from ReviewPlus IDL Application

On the Master, when we choose the “Set Signals” item from menu “Edit”, the event handler “ReviewPlus_SignalDialog_event” is invoked, and the tasks in the handler are processed. In it, we can put statements to get information of the event structure and put pieces of substrings for its fields in a message string, as we have described in Section 3, Shared Event Model. We should also put a substring for locating the desired event handler, i.e., “Edit>SetSignals” followed by a delimiter in the event message string for NaradaBrokering to broadcast.

On the participant, it receives this string from the public variables in the polling structure. After it parses and gets the substrings, it locates the event handler and rebuilds the event structure in IDL types. It then calls the event handler with the event structure like this:

```
ReviewPlus_SignalDialog_event,
{WIDGET_BUTTON, ID:15, TOP:1, HANDLER:15,
SELECT:1}
```

6. Future Work

We are working on the ReviewPlus package to make it collaborative between computers using the polling structure. It is a big package. We plan to finish the implementation of the collaboration to make the Master and Participant clients available in real life and daily use. We are also interested in finding new and better structures, methodologies and technologies for IDL applications to be collaborative over networks, platforms and environments.

7. Conclusion

In this paper we have described the Grid-base collaboration paradigm, the shared event model, and the common message broker that together enable effective collaboration between computers over the Internet to be possible. We have focused on the design of making IDL applications collaborative. To do that, we have proposed the Notifying structure and Polling structure, and analyzed the advantages and limitations of each of them. We are finally working on a real life and useful IDL application (ReveiwPlus) to make it collaborative, taking the advantage of the Polling structure. We are also working to find new and better ways to achieve high performance collaboration in general IDL applications.

References

[1] Liam E. Gumley, *Practical IDL Programming*, Morgan Kaufmann Publishers, San Francisco, CA 94104-3205, USA, 2002.
[2] Research Systems Inc. http://www.rsinc.com/
[3]Fanning Consulting
http://www.dfanning.com/documents/idllinks.html
[4] Geoffrey Fox, Shrideep Pallickara, and Xi Rao, "A Scalable Event Infrastructure for Peer to Peer Grids", proceedings of 2002 Java Grande/ISCOPE Conference, Seattle, November 2002, ACM Press, ISBN 1-58113-599-8, pages 66-75. http://grids.ucs.indiana.edu/ptliupages/publications/ScaleableEventArchForP2P.doc
[5] Fran Berman, Geoffrey Fox, and Tony Hey, *Grid Computing: Making the Global Infrastructure a Reality*, John Wiley & Sons Ltd, Chichester, West Sussex PO19 8SQ, England, 2003.
See http://www.grid2002.org
[6] ReviewPlus Data Visualization Software User Manual
http://web.gat.com/comp/analysis/uwpc/reviewplus/manual/
[7] General Atomics and Affiliated Companies http://www.ga.com/
[8] Sheng Liang, *The Java Native Interface*, Addison-Wesley, Sun Microsystems, Inc., Palo Alto, CA 94303, USA, 1999.
[9] MDSplus: Introduction
http://www.mdsplus.org/intro/index.shtml